CMPT 280

Topic 21: Travelling Salespeople, Greedy Algorithms, and Backtracking

Mark G. Eramian

University of Saskatchewan

References

- Textbook, Chapter 21

The Traveling Salesperson Problem (TSP)

- Suppose a salesperson
  - Starts at home
  - Visits a number of locations
  - Returns home
- There are many possible routes to take (permutations of intermediate locations).
- Objective: find the best route (minimize distance, time, cost, etc).

TSP Formulation

- TSP is normally formulated as a problem on a weighted undirected graph $G = (V, E)$...
  - Nodes in V represent locations to be visited, including "home".
  - There is an edge $[u, v] \in E$ if it is possible to travel directly between locations u and v.
  - The weight on an edge represents the cost of taking that edge.
- Problem: Find the shortest Hamiltonian path (or cycle, sometimes called a tour).
- Naive solution: test all Hamiltonian paths; pick the smallest cost. $O(n!)$ algorithm!!!

Solving Optimization Problems

General approaches:

- Greedy Algorithms
- Backtracking

Solving Optimization Problems

Greedy Algorithms

- Greedy algorithms:
  - Start with a partial solution.
  - Repeatedly extend the partial solution until a complete solution is obtained.

- A partial solution is *feasible* if it is anticipated that it can be extended to a complete solution that satisfies all of the constraints.

- The Greedy Approach:

```
1  Start with a (usually trivial) partial solution
2
3  While the solution is not complete
4      Extend the partial solution to a more complete solution in a way that
5      seems like it should lead to a good solution while maintaining
6      feasibility.
```

Solving Optimization Problems

Greedy Algorithms

- For a given problem there can be many different greedy algorithms corresponding to:
  - Different ways to define what form a partial solution takes
  - Different choices for the initial partial solution
  - Different ways of doing an extension

Solving Optimization Problems

Example: find the shortest path from s to t.

Unweighted Graph:

- Partial Solution Form: partial shortest-paths tree from s
- Initial partial solution: the start vertex s
- Extension: Select an edge with
  - One end already in the tree.
  - The other end not in the tree.
  - The closest endpoint to s as possible.

  (This is exactly the breadth-first search algorithm for finding shortest paths, guarantees optimal solution)

Solving Optimization Problems

Example: find the shortest path from s to t.

Weighted Graph:

- Partial Solution Form: partial shortest-paths tree from s
- Initial partial solution: the start vertex s
- Extension: Select an edge with
  - One end already in the tree.
  - The other end not in the tree.
  - The path from s to the non-tree edge vertex is as short as any other path from s to a non-tree vertex.

  (Dijkstra's algorithm, guarantees optimal solution)

Solving Optimization Problems

Example: find the shortest path from s to t.

Alternate solution for Weighted Graphs:

- Partial Solution Form: partial shortest-paths tree from s
- Initial partial solution: the start vertex s
- Extension: Select an edge with
  - One end already in the tree.
  - The other end not in the tree.
  - Has the smallest edge weight.

  (This is actually the minimum spanning tree algorithm!)
- Produces a good, but non-optimal solution to the shortest paths problem.
- Greedy algorithms do not necessarily find the optimal solution!!!

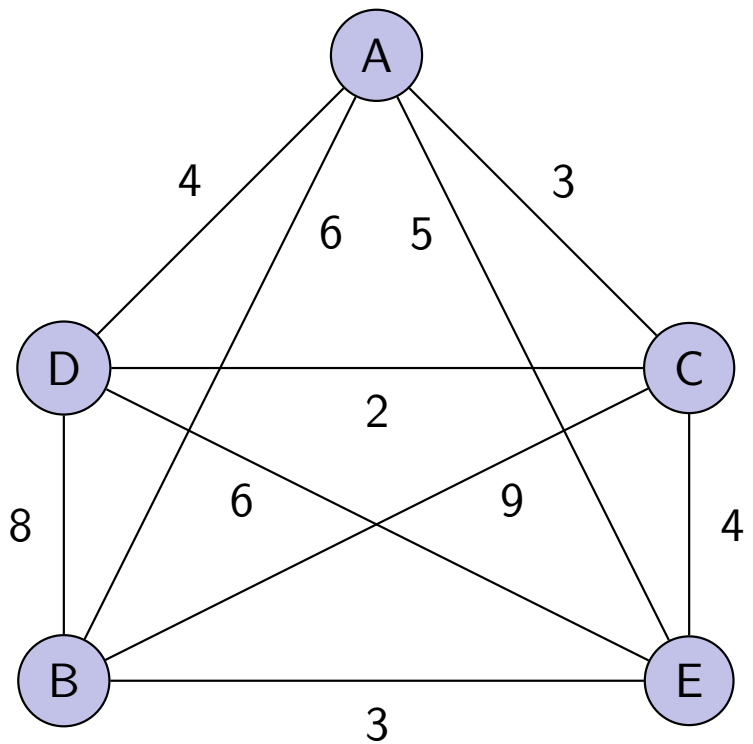
Solving TSP

A Greedy Approach

- Let s be the start vertex ("home");
- Partial solution: A path (no repeated vertices) from s to some vertex x.
- Extension: If the path contains all vertices, add the edge from x back to s (algorithm fails if this edge does not exist), otherwise, select a vertex v adjacent to x and add it to the end of the path.
- What if there is more than one choice for v? – use greedy approach!
  - select the vertex not already on the path (feasible) which has the smallest cost edge to x (greedy)

Exercise 1

Use the greedy algorithm on the next (or previous) slide to find a TSP solution for the following graph. Is the solution we get optimal?



Solving TSP

A Greedy Approach

Greedy #1 algorithm:

```
1   Select a start vertex s
2
3   // Partial tour is a sequence of edges that forms a path from start to cur.
4   tour = empty
5   cur = start
6
7   set all vertices as unreached
8   set cur as reached
9
10  while tour has less than n-1 edges (i.e., < n vertices)
11      find the minimum weight edge from cur to a vertex, u, that is unreached
12      add [cur, u] to the end of the tour
13      cur = u
14
15  add edge [cur, start] to the end of the tour
```

Time complexity = ??

Solving TSP

A Greedy Approach

- Time for steps before the loop: $O(n)$, $n = |V|$.
- Number of loop iterations: $n - 1$ (0 to $n - 2$ edges).
- Cost for each time through loop: $O(degree(cur))$ (assuming adj. list implementation)
- Time for steps after the loop: $O(1)$

Total time is:

$$n + \sum_{cur} degree(cur) = n + O(m) = O(n + m) = O(|V| + |E|)$$

where m $= |E|$.

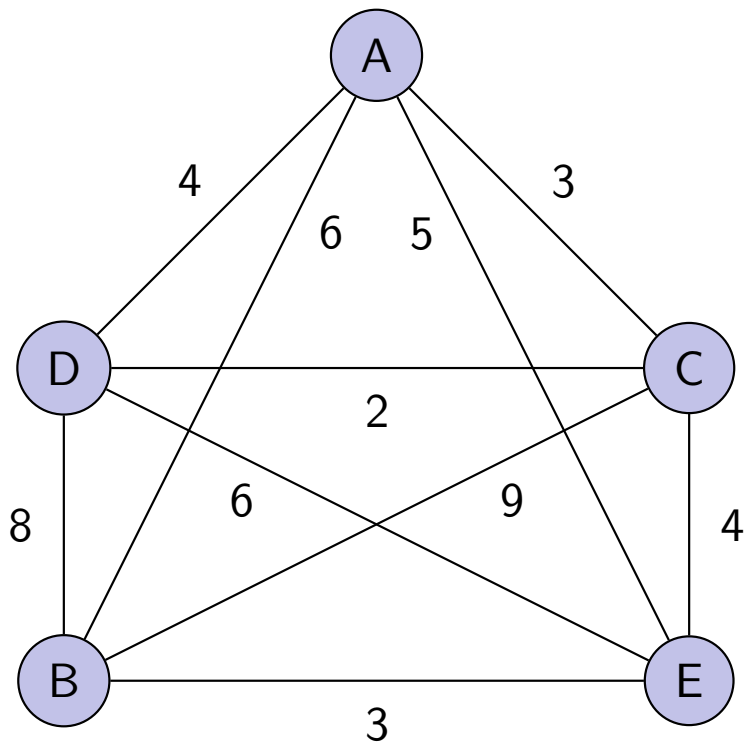
Solving TSP

Alternate Greedy Approach

- Partial solution: A set of edges (initially empty) where there are no more than 2 edges incident on one vertex and no cycles of length $< n$.
- Extension: Add the minimum length feasible edge.

Exercise 2

Use the modified greedy algorithm on the previous slide to find a TSP solution for the following graph. Is the solutions we get optimal?



Solving TSP

Greedy Approach: Summary

- Greedy approach may not find the optimal tour.
- Usually polynomial time (otherwise, why settle for a potentially sub-optimal solution!).
- Greedy approach may not even find a feasible tour **at all**, even if one exists! (previous algorithm fails if edge $[d, b]$ is removed).
- It is believed that no algorithm exists that both
  - Uses the greedy approach
  - Always finds the optimal solution to TSP.

Solving TSP

Greedy Approach: Summary

- Further it is known that TSP is NP-Hard.
- Decision version is NP-Complete (is there a tour with cost $\leq C$).
- NP-Complete problems are a class of problems for which it is **believed** that there is no algorithm which:
  - always finds the optimal solution; **and**
  - executes in time $O(n^k)$ for some k.
- If there were such an algorithm for TSP, this would imply that P=NP.
- NP-complete problems are not believed to have algorithms that run in polynomial time on an electronic computer. (With unlimited parallelism, then can run in polynomial time, but no such machine exists).

Backtracking

- *Backtrack programming* is:
  - Another approach to algorithm development
  - Guaranteed to find the optimal solution to optimization problems.
  - computationally less efficient (usually exponential time)
- Basic approach: try **ALL** alternatives in an organized manner.
- Improvements over trying every possible solution:
  - Eliminates non-feasible partial solutions
  - Eliminates partial solutions that can never lead to an optimal solution (bounding)
- Even with these improvements, backtracking usually requires exponential time ($O(a^n)$ for some constant real a).

Backtracking

General Approach

```
1   driver method:
2       best solution = nil
3       T = initial partial solution
4       testAllExtensions(T)
5       output the best solution
6
7   method testAllExtensions (T):
8       if T is a complete solution
9           if T is better than the current best solution
10              save T as the best solution
11      else
12          for each possible extension of T
13              apply the extension to T
14              if feasible(T) && T has potential to be better than current best
15                  testAllExtensions (T)
16              remove the extension from T
```

- "For each possible extension of T" tries all the alternatives for a one step extension of the current partial solution T.
- For a feasible extension, all further extensions are tried recursively.
- After trying an extension, remove the extension before trying the next one.

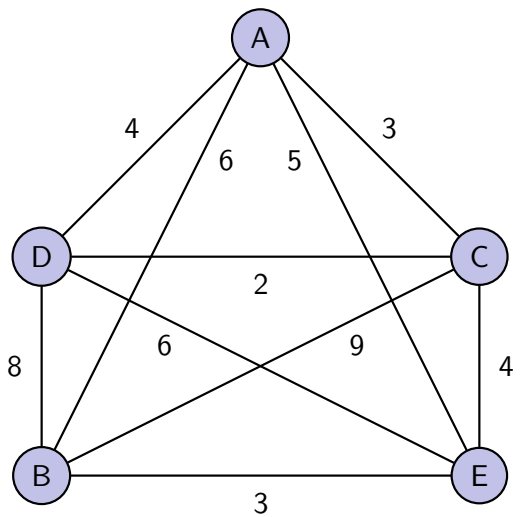
Backtracking

Applying Backtracking to TSP

- A partial solution is a path from start vertex s to to some vertex x.
- Initial partial solution: s (or empty edge list)
- Solution complete when: n vertices on path (or list of $n-1$ edges), still need to add last edge back to s.
- Feasible: extend only to unvisited nodes
- Potential to be better than current best: if cost of partial path + estimate of cost for remaining edges $<$ value of best tour.
  - Note: estimate for cost of remaining edges must be a lower bound estimate (better estimates will result in more bounding)
  - Possible estimates:
    - 0
    - number of remaining edges $\times$ min cost of any edge
    - $\sum_{\text{unreached vertices v}}$ least cost edge incident on v

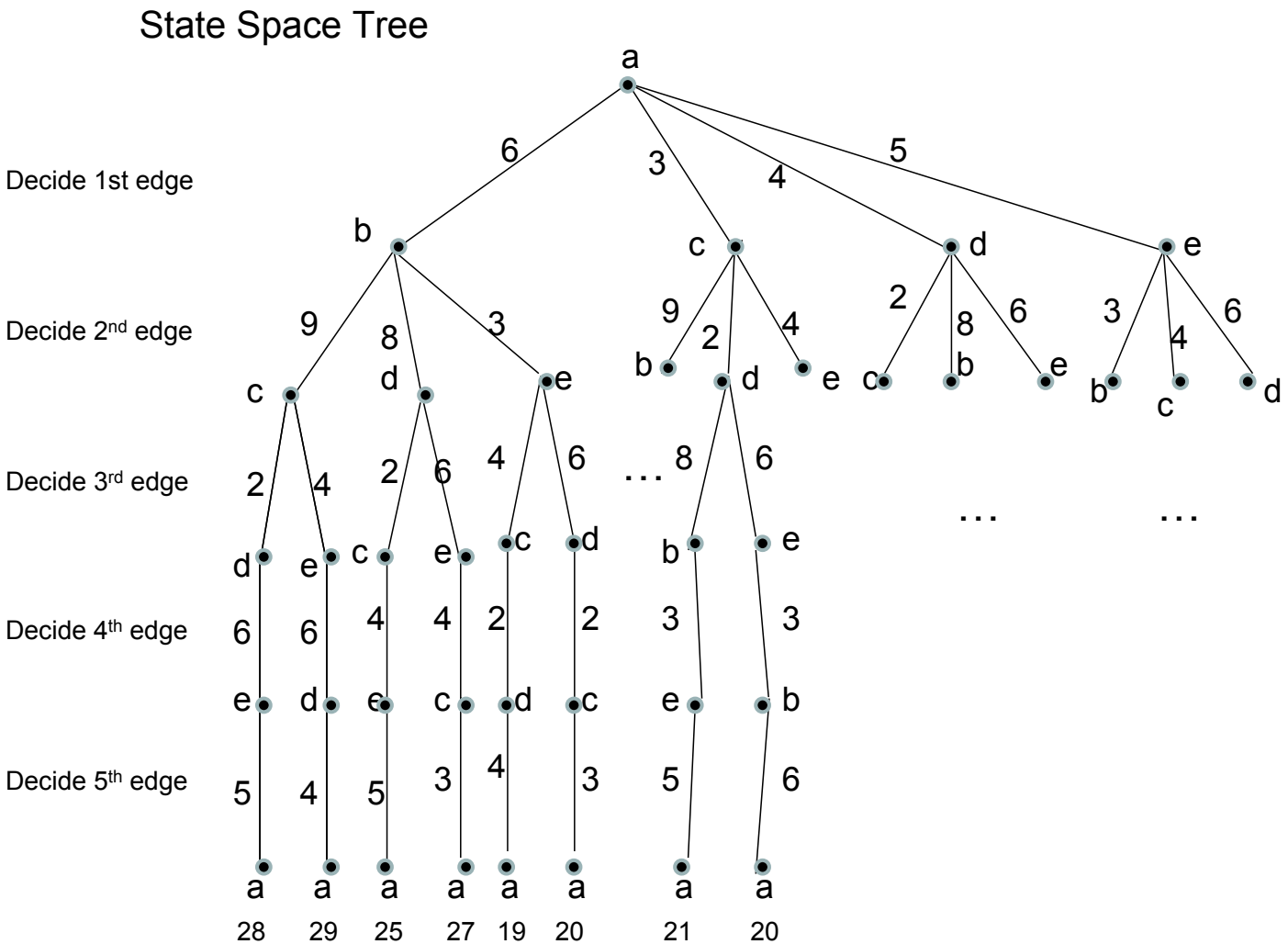
View of Backtracking

- Extending a partial solution can be viewed as making a decision.
- The same decision is made for each extension, but decision reflects the current partial solution.
- For the TSP: the i-th extension equates to deciding which edge is to be the i-th edge of the tour.
- A *state-space tree* can be used to display the alternatives for each decision.



TSP State Space Tree

(Non-feasible solutions already omitted)



TSP State Space Tree

- State space tree represents all possible partial solutions.
- Branching factor at each level is reduced by one (one less edge to pick from).
- Height of tree is $n-1$, but there are $(n-1)!$ leaves.
- Adding bounding will significantly reduce the number of alternatives – some paths will not be followed.
- On average, bounding tends to make the branching factor close to some constant a, and we end up with a tree with about a^n leaves. This is why backtracking tends to result in exponential time algorithms.

TSP State Space Tree

- Backtrack searching corresponds to a depth first search of the state space tree.
- Other searches on the state-space tree correspond to other algorithmic techniques:
  - Breadth first corresponds to a technique called *FIFO branch-and-bound*.
  - Bottom-up search corresponds to *dynamic programming*.